Data Ingestion from a Data Lake: The Case of Document-oriented NoSQL Databases

Fatma Abdelhedi[1], Rym Jemmali[1,2] and Gilles Zurfluh[2]
[1]*CBI², Trimane, Paris, France*
[2]*IRIT CNRS (UMR 5505), Toulouse University, Toulouse, France*

Keywords: Data Lake, Data Warehouse, NoSQL Databases, Big Data.

Abstract: Nowadays, there is a growing need to collect and analyze data from different databases. Our work is part of a medical application that must allow health professionals to analyze complex data for decision making. We propose mechanisms to extract data from a data lake and store them in a NoSQL data warehouse. This will allow us to perform, in a second time, decisional analysis facilitated by the features offered by NoSQL systems (richness of data structures, query language, access performances). In this paper, we present a process to ingest data from a Data Lake into a warehouse. The ingestion consists in (1) transferring NoSQL DBs extracted from the Data Lake into a single NoSQL DB (the warehouse), (2) merging so-called "similar" classes, and (3) converting the links into references between objects. An experiment has been performed for a medical application.

1 INTRODUCTION

The last few years have seen an explosion of data generated and stored by a large number of computing devices. The resulting databases are referred to as "Big Data", which refers to the so-called "3V" rule: volume, variety, and velocity. This rule characterizes data lakes (Couto et al., 2019) that can bring together several databases of different types and formats such as relational databases, NoSQL databases (Not only SQL), CSV files, text files and spreadsheets. This massive and complex data represents an essential reservoir of knowledge for decision-makers; however, the volume of data as well as the diversity of structures and formats constitute a major obstacle to decision processing. In order to facilitate the decision analysis of a data lake, we propose to integrate the data in a NoSQL (Not only SQL) data warehouse. It should be noted that the problem statement presented in this article has been voluntarily restricted to the ingestion of the only document-oriented NoSQL DBs contained in the Data Lake; neither is the selection of the data in the Data Lake treated, i.e., the totality of each DB is transferred into the Data Warehouse. Our work is part of a medical application for health insurance companies.

The remainder of the paper is structured as follows: Section 2 presents the medical application that justifies the interest of our work. Section 3 reviews the state of the art. Section 4 presents our contribution which consists in ingesting the DBs of a Data Lake into a NoSQL Data Warehouse. Section 5 details the development of our prototype and describes the experimentation of our process. Finally, Section 6 concludes the paper and announces future work.

2 CASE STUDY AND PROBLEM STATEMENT

Our work is motivated by a project developed in the health field for a group of mutual health insurance companies. These insurance companies, stemming from the social and solidarity economy, propose to their customers a coverage of the medical expenses which comes in complement of those refunded by a public institution: the health insurance fund. To ensure the management of their clients, these mutual health insurance companies are faced with a significant increase in the volume of data processed. This is all the more true since some of these companies act as social security centers, i.e., clients

Abdelhedi, F., Jemmali, R. and Zurfluh, G.
Data Ingestion from a Data Lake: The Case of Document-oriented NoSQL Databases.
DOI: 10.5220/0011068300003179
In *Proceedings of the 24th International Conference on Enterprise Information Systems (ICEIS 2022) - Volume 1*, pages 226-233
ISBN: 978-989-758-569-2; ISSN: 2184-4992

transmit the entire reimbursement file to a mutual insurance company, which performs all the data processing related to the file. In order to monitor these files, numerous exchanges are necessary between the health insurance fund, the mutual insurance companies and the health professionals. Under the impetus of the State, the health insurance fund has developed a Digital Health Space[1] (ENS). The ENS stores the medical data of each insured person. This data is confidential in nature, but the holder may authorize a health professional or institution to consult or add to his ENS. All access and actions carried out on the ENS by non-holders are traced. Mutual health insurance companies therefore have limited access to the ENS from which they can extract data in order to process the files of their policyholders and, more generally, to carry out analyses of any kind (in compliance with confidentiality rules). For each insured person, the ENS contains administrative data, medical records (measurements, medical imaging archives, reports, therapeutic follow-up, etc.), reimbursement history and questionnaires. When the ENS is fully deployed at the national level, its volume will be considerable since it concerns 67 million insured persons.

In the context of this project, the ENS constitutes a real Data Lake because of (1) the diversity of data types, media, and formats (2) the volumes stored which can reach several terabytes and (3) the raw nature of the data. The objective of the project is to study the mechanisms for extracting data from the ENS and organizing it to facilitate analysis (Big Data Analytics). Our work aims to develop a system that allows insurance companies to build a data warehouse from an existing data lake. In this paper, we limit the scope of our study as follows:

- The ENS Data the source of our process and contains massive structured and unstructured data. In this article, we limit ourselves to document-oriented NoSQL databases managed by the MongoDB system. Indeed, this category of datasets represents an important part of the ENS data. The Data Lake contains various datasets concerning in particular the files of the insured, the activity of the doctors, the medical follow-ups.
- The Data Warehouse generated by our process is a document-oriented NoSQL DB;at this stage of our study, the Data Warehouse is not organized according to a multidimensional model. Moreover, although MongoDB is the most widely usedNoSQL system in the industry today (DB- Engines Ranking, s. d.) we chose OrientDB [2] to manage the Data Warehouse. Indeed, this NoSQL system offers advanced functionalities, notably the possibility of expressing several types of semantic links. Itwill thus be easier to restructure the data of the Data Warehouse to facilitate decisional treatments (the works on decisional treatments are not discussed in this article).

3 RELATED WORKS

Our study consists in extracting data from a Data Lake containing several MongoDB databases and then integrating them into a NoSQL Data Warehouse. In this section, we review existing works that address this problem.

The papers (Kuszera et al., 2019; Liyanaarachchi et al., 2016; Mahmood, 2018), have proposed techniques to transfer a relational database into a document-oriented NoSQL DB. In particular, the proposed algorithms transform foreign keys into nested data. In (Mallek et al., 2018), the authors propose to transfer data from a NoSQL DB into a multidimensional warehouse using MapReduce to accelerate the data transformation. Furthermore, the work (Maity et al., 2018) specify a generic process to transfer data from a NoSQL DB (especially

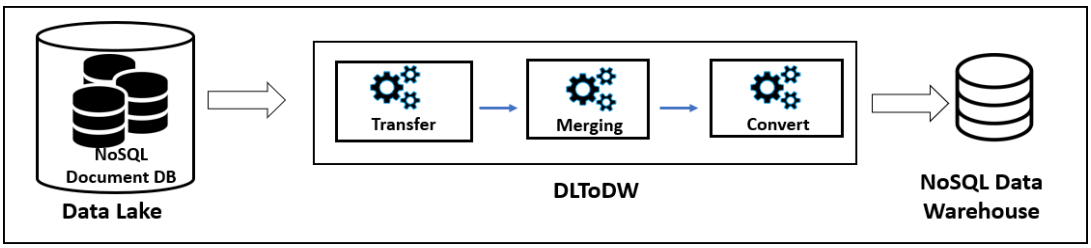

Figure 1: Overall architecture of the DLToDW data lake extraction process.

[1] https://gnius.esante.gouv.fr/reglementation/fiches-reglementation/mon-espace-sante

[2] https://orientdb.org/

document-oriented or graph-oriented) to a relational DB. This type of transfer allows benefiting from the functionalities of relational data management. Conversely, the authors in (Yangui et al., 2017) used Talend software to implement rules for transforming a multidimensional data model into a document-oriented NoSQL DB. This takes advantage of the Big Data context and in particular of the MapReduce operation. Facts are stored in a table and dimensional links are translated by nesting dimensions into facts. Finally, the authors in (Dabbèchi et al., 2021) proposed a process to integrate social media data in a NoSQL Data Warehouse. The data is extracted from a data lake containing column-oriented and document-oriented NoSQL databases. The paper defines matching rules to transfer this data into the NoSQL warehouse.

The works, which we have just presented, have proposed mechanisms allowing the transfer of data between datasets of different types. Thus, the articles (Kuszera et al., 2019; Liyanaarachchi et al., 2016; Mahmood, 2018) integrate a relational DB into a NoSQL DB. Other works (Mallek et al., 2018; Maity et al., 2018; Yangui et al., 2017) ensure the transfer from a NoSQL DB to a multidimensional warehouse or vice versa. Finally, the process presented in (Dabbèchi et al., 2021) allows the integration of NoSQL DBs into a NoSQL data warehouse. Overall, this state of the art shows the difficulty of exploiting complex and massive data and the need to reorganize these data in a form adapted to decision analysis. The proposed processes mainly concern the integration of data in NoSQL systems which offer interesting object functionalities for decision support. However, these solutions only partially solve our problem as our medical case study includes specificities that are not resolved in the proposed processes. Thus, in our application, "similar" data belonging to different datasets must be merged. Moreover, the numerous semantic links between objects cannot only be translated into nested data structures.

In this paper, we propose a new automatic approach to ingesting document-oriented NoSQL DBs into a NoSQL data warehouse by supporting "similar" data merging and link conversion.

4 THE DLToDW INGESTION PROCESS

Our objective is to automate the ingestion of several NoSQL databases present in a Data Lake; the extracted data are transferred to a NoSQL Data Warehouse. We have developed the DLToDW process that successively transfers the data from the Data Lake to the target DB (the Data Warehouse), converts the semantic links in the target and merges the so-called "equivalent" objects. Figure 1 shows the three modules of our process.

The I/O of our process, i.e., the NoSQL DBs contained in the Data Lake and the NoSQL Data Warehouse, follow the same document-oriented model. The metamodel describing these databases is presented in Figure 2 and is formalized with the UML[3] class model. The rectangles describe object classes, and the arcs represent association, composition, and inheritance links.

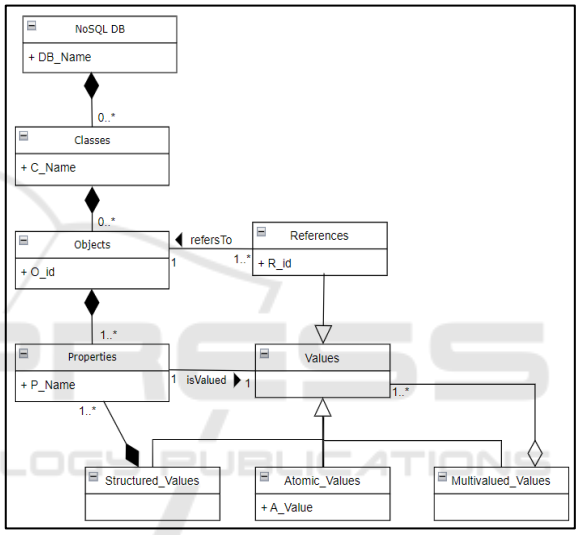

Figure 2: Meta model of a document-oriented NoSQL database.

This metamodel shows that any class groups objects (also called Document or Record). Each object has an identifier and contains a set of properties in the form of couples (Name, Value). A value can be atomic, multivalued, or structured.

In the following sections, we describe the DLToDW process by presenting the three modules that comprise it.

4.1 The Transfer Module

This process transfers the DBs from the data lake to a single data warehouse. The Data Lake, the source of the transfer, normally contains data of different formats such as relational DBs, NoSQL DBs or CSV files; however, in this article we have restricted the

[3] https://www.omg.org/spec/UML/2.5.1/About-UML/

source to document-oriented NoSQL DBs. The target of the transfer is also a document-oriented NoSQL DB.

In our case study presented in Section 2, the (restricted) Data Lake consists of MongoDB NoSQL DBs and the Data Warehouse is an OrientDB NoSQL DB. MongoDB and OrientDB share the same document-oriented model using different vocabularies; however, the underlying concepts in terms of data representation are identical and consistent with the metamodel in Figure 2. Note that OrientDB offers a richer description capability, especially in terms of link expression. For the sake of clarity of presentation, we will use the MongoDB vocabulary for the source and the OrientDB vocabulary for the target (Table 1).

Table 1: Correspondence of the source and target vocabularies.

MongoDB (source)	OrientDB (target)
Collection	Class
Document	Record
Fields	Property
DBRef link	Reference
Oid (Object identifier)	Rid (Record identifier)

The Transfer module therefore consists of "copying/pasting" data from several databases (source) into a single database (target). This data transfer must however follow certain rules to allow the treatments that we will carry out later.

Rule 1: the names of the transferred classes are prefixed by the name of the source database from which they originate; this avoids the synonymity of class names in the data warehouse.

Rule 2: when transferring documents, the original identifiers are kept as they are; thus, the identifier of a document is stored in a property of the target record, in the form (Oid, value of the identifier). This identifier will be used in the Convert module andthen deleted in the record.

Rule 3: the links contained in the documents (DBRef links) are stored as they are in the target records; theywill be transformed into references in the Convert module. The transferred links will be prefixed by thekey word "DBRef".

According to rule 2 above, any record r that is stored in the Data Warehouse has a property containing the MongoDB identifier of the original document. If we consider the set of OrientDB identifiers (E_Rid) assigned to the records and the set of MongoDB identifiers (E_Oid) present in the source, then there exists a bijection of E_Oid into E_Rid that we will note as follows: Rb: E_Oid → E_Rid. This property is important because it allows MongoDB links to convert into OrientDB links.

4.2 The Merging Module

The Data Lake is generally made up of a set of separate databases managed independently. The data warehouse resulting from the Transfer module may contain "similar" data sets. For example, in our medical application, descriptions of insured persons or lists of doctors appear in several databases in the Data Lake. These data may concern the same entities in the real world but do not necessarily have the same structures (different properties).

The Data Warehouse can therefore include subsets containing classes linked by the equivalence relation; they "describe the same entities"; each subset is called an equivalence group. For example, in the ENS Data Warehouse of our application, the classes B1.Doctors, B2.Therapists and B3.Practitioner constitute an equivalence group (the prefixes B1, B2 and B3 correspond to the names of the original databases in the Data Lake).

For each equivalence group in the data warehouse, the module will cause all classes in the group to be deleted and replaced by the resulting class X. The transformation from G to X is based on the use of a domain ontology created with the help of business experts; the ontology automatically builds the X class. The experts (administrators, managers, decision-makers, etc.) have in-depth knowledge of one or more databases contained in the Data Lake.

They are asked to specify the possible semantic correspondences between the data of the different databases. For example, in the ENS Data Lake, three databases contain data describing individuals insured by mutual insurance companies. The ontology will indicate an equivalence relation between these data. The similarity relations between data are stored in a domain ontology, called "Onto", in the form of a graph; they are obtained from the specifications provided by the business experts.

The business experts must define a primary key (in the sense of the relational model) among the attributes of the classes of the same group. This key, made up of a minimal set of properties, makes it possible to distinguish records and to establish inter-class correspondences within a group. For example, the classes of a group containing descriptions of

doctors have the national professional identification number of doctors as a key. This key allows records about the same real-world entity to be grouped. In addition, the experts examine the other properties present in the classes and indicate whether it is appropriate to keep them in the data warehouse (Properties class). These properties are transferred by the Merging module to the resulting X class.

To transform the classes of a group into a single class X, a transfer of values from the group to X must be specified. Thus, for each property to be transferred, the experts indicate in the ontology the class of the group that is the source of the transfer and that is likely to ensure optimal data quality; this is the class of the group that is considered by the experts as the most reliable. This source may vary for different properties of X.

The Merging module of our process aims at substituting each group of equivalence classes by a unique class. We have the "Onto" ontology and the Data Warehouse (DW) at our disposal. The ontology describes the composition of each group and the characteristics of the resulting class X. The Data Warehouse contains the classes that will be merged by the Merging module. This module produces a new version of the NoSQL Data Warehouse on which the Convert processing is applied, responsible for transforming DBRef type links (specific to MongoDB) into links in the form of references between objects (present in OrientDB).

4.3 The Convert Module

The MongoDB system offers the possibility of establishing links between documents in collections; the links are expressed by means of structured data marked with the DBRef reserved word. This form of linkage is compatible with all document-oriented NoSQL systems insofar as these systems accept structured properties and referential integrity is not assured. A DBRef link is monovalent or multivalent; it can be located at the first level of the properties of a collection or nested in a structured property. For the needs of our case study, which requires good data quality, we have chosen to transform these links in the data warehouse. We use references between objects with referential integrity control according to the principles of the object standard defined by the ODMG[4]. This reference mechanism is available in the OrientDB system.

In a DB, a link is a relationship that maps an object (record or document) to one or more other objects. During the transfer of documents, each MongoDB link was stored as is (without transformation) in the form of a property in a record in the data warehouse, in accordance with rule 3 applied by the Transfer module (see section 4.1). At this stage, the links cannot be used in the data warehouse. The Convert module is therefore responsible for substituting each MongoDB link with an OrientDB link.

Before the execution of the Convert module: among the properties of any record, one or more MongoDB links can be found. Such a link-property is characterized by a structured value containing the DBRef keyword and a name giving the link semantics.

The Convert module will transform each MongoDB link into an OrientDB link thanks to the bijective application Rb between E_Oid and E_Rid. The value of the MongoDB link $Oid \in E\_Oid$ Oid will be substituted by the $Rid \in E\_Rid$ of the Data Warehouse record in which the MongoDB identifier is located.

5 PROTOTYPE DEVELOPMENT AND EXPERIMENTATION

To implement the DLToDW process, we have developed a prototype allowing the ingestion of several NoSQL databases according to the principles outlined above. Our process includes the three modules Transfer, Merging and Convert which are implemented successively. We have chosen the OMG[5] Model Driven Architecture (MDA) to develop the Transfer module. The two modules Convert, and Merging are based on algorithmic processing coded in Java. We implemented our prototype with the data from the medical application presented in Section 2.

5.1 Data Transfer

We used the MDA to develop the Transfer module because this component is part of the production of a larger software system not described in this article. Indeed, our case study is based on a Data Lake containing, in addition to the NoSQL DBs presented here, relational DBs, Json files, and spreadsheets. In this application framework, MDA offers on the one hand a great flexibility for the development: modeling of inputs and outputs and expression of model

[4] http://www.odbms.org/odmg-standard/

[5] The Object Management Group (OMG) https://www.omg.org

transformation rules. On the other hand, it facilitates the development and maintenance of the software by ensuring the generalization of the processes developed: the same principles are applied to each data set. According to the MDA approach, software development consists of describing separately the functional specifications and the implementation on a platform. It is based on three models representing the levels of abstraction of the application: (1) the requirements model (CIM for Computation Independent Model) in which no IT considerations appear, (2) the analysis and design model (PIM for Platform Independent Model) independent of the technical details of the execution platforms and (3) the code model (PSM for Platform Specific Model) specific to a particular platform. Our work is located at the PSM level where the physical schemas are described. In our application, the principles of MDA consist in modeling the input and output of a process and in providing a set of transformation rules applied to the models to go from input to output. In order to preserve a certain generality of the treatments, the input and output models are represented by metamodels.

The Transfer module has been translated into two metamodels (one input and one output) and a set of model transformation rules expressed in the declarative language QVT[6] (Query/View/Transform) specified by the OMG. We used the EMF[7] (Eclipse Modeling Framework) development environment with the Ecore metamodeling language. The input (MongoDB) and output (OrientDB) metamodels expressed in Ecore are consistent with the UML diagram in Figure 2. Ecore relies on XMI to instantiate the models.

This module therefore transfers the NoSQL databasesfrom the Data Lake into a warehouse which is a single NoSQL database. At the end of the processing, the warehouse contains a set of classes with the following properties:

- The number of classes in the warehouse is equal to the sum of the number of collections contained in the Data Lake (all databases combined)
- Each class takes the name of the original collection prefixed by the name of the DB in the Data Lake

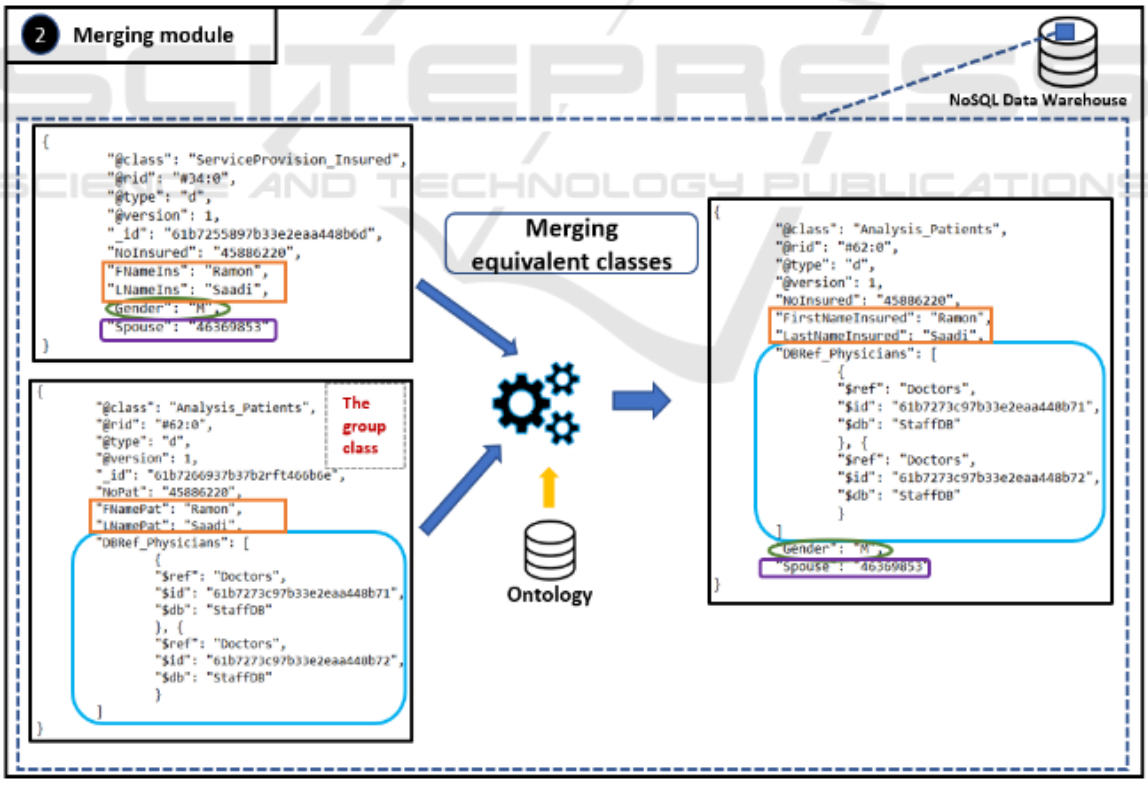

Figure 3: Data flow and processing in the Merging module.

[6] https://www.omg.org/spec/QVT/1.2/PDF

[7] https://www.eclipse.org/modeling/emf/

- Each record of a class has an identifier (Rid) assigned by the system managing the warehouse (OrientDB)
- The links contained in the warehouse records have been stored in their original format (MongoDB DBRef links).

5.2 Merging of Similar Classes

In a second step, the Merging module is applied to the warehouse and its objective is to merge the classes considered as "similar". To perform the merging, this module uses a domain ontology[8] provided by experts of our case study. This ontology has been stored in a PostgreSQL[9] relational database. For example, the ontology indicates that the classes "ServiceProvision_Insured"and "Analysis_Patients" belong to the same equivalence group. These two classes come from two distinct DBs ("ServiceProvision" and "Analysis") in the Data Lake. The ontology provides all the properties of the unique class X elaborated by the Merging module and specifies the class from which these properties will be extracted. In the case where "similar" properties (for example the Name of the insured) are found in several classes, the ontology indicates the class that will be chosen to extract the values and save them in X. The class X is the result of the merging of several classes; its name, which appears in the ontology, was assigned by the experts. Figure 3 shows the merging of 2 records in the data warehouse.

5.3 Link Conversion

The Transfer module has stored all the links in the warehouse in their original format (MongoDB DBRef link). The Convert module will transform each of these links into Rid references between records; these links in the form of references conform to the object principles set out by the ODMG. This conversion of links requires a complete scan of the warehouse. In fact, we did not think it wise to perform this treatment when the warehouse is fed (Transfer module) because:

1. the dispersion of referenced records (which requires that all classes be created at the beginning of the process)
2. the number of links per record; in our application, every record contains at least one link. For example, in Figure 4, the "DBRef_Physicians" field of the "Analysis_Patients" Class corresponds to a multivalued value of DBRef, so each DBRef structure will be replaced by the Rid of the referenced record of the "Doctors" collection from the "StaffDB".

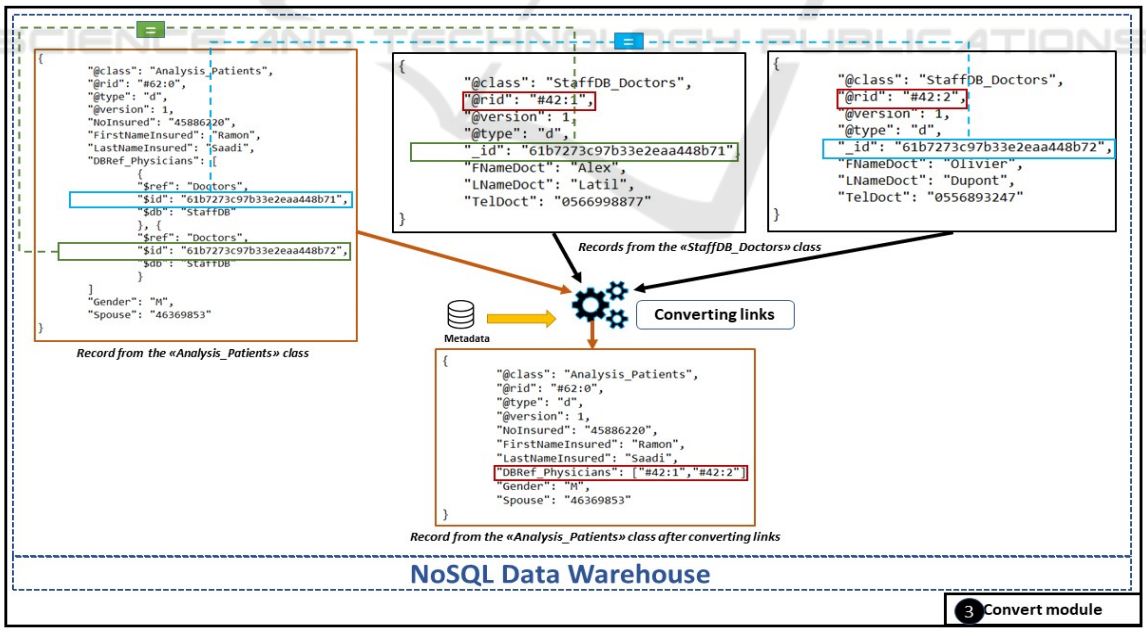

Figure 4: Data flow and processing in the Convert module.

[8] The principles of exploitation of the ontology are not detailed inthis article

[9] https://www.postgresql.org/

6 CONCLUSION

We have proposed a process for ingesting data from a Data Lake to a Data Warehouse; the latter consists of a single NoSQL DB while the Data Lake contains several DBs. Thus, the article by (Yangui et al., 2017) deals with the transformation of a multidimensional DB into a NoSQL document data warehouse and does not consider specific links to MongoDB (DBRef). On the other hand, the work (Dabbèchi et al., 2021) proposes a process to transfer data stored in NoSQL databases (the Data Lake) and feed a NoSQL Data Warehouse. The links between objects are not materialized in the target as references. In order to focus this paper on a limited problem, we have limited the content of the Data Lake to document-oriented NoSQL DBs. Three modules were implemented successively to ensure the ingestion of data. The Transfer module transformsthe Data Lake databases into a NoSQL database; its development is based on the MDA. The Merging module merges classes considered semantically equivalent and belonging to different Data Lake databases; this merger is based on an ontology provided by business experts. Finally, the Convert module translates the links in MongoDB into references according to the principles of object databases supported by the OrientDB system. Our process has been implemented in an application for health professionals. Currently, we are extending the DLToDW process both upstream (the Data Lake) and downstream (the Data Warehouse). In particular, we are defining a generic model and approach to integrate the other types of datasets that make up the Data Lake.

REFERENCES

Couto, J., Borges, O., Ruiz, D. D., Marczak, S., & Prikladnicki, R. (2019). *A Mapping Study about Data Lakes : An Improved Definition and Possible Architectures. 453-458.*

Dabbèchi, H., Haddar, N., Elghazel, H., & Haddar, K. (2021). *Social Media Data Integration : From Data Lake to NoSQL Data Warehouse 701-710.*

DB-Engines Ranking. (s. d.). DB-Engines. *https://db-engines.com/en/ranking/document+store*

Kuszera, E. M., Peres, L. M., & Fabro, M. D. D. (2019). *Toward RDB to NoSQL : Transforming data with metamorfoseframework. Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, 456-463.*

Liyanaarachchi, G., Kasun, L., Nimesha, M., Lahiru, K., & Karunasena, A. (2016). *MigDB - relational to NoSQL mapper. 2016 IEEE International Conference on Information and Automation for Sustainability (ICIAfS), 1-6.*

Mahmood, A. A. (2018). *Automated Algorithm for Data Migration from Relational to NoSQL Databases. Al-Nahrain Journal for Engineering Sciences, 21,60-65.*

Maity, B., Acharya, A., Goto, T., & Sen, S. (2018). *A Framework to Convert NoSQ L to Relational Model. ACIT 2018: Proceedings of the 6th ACM/ACIS International Conference on Applied Computing and Information Technology, 1-6.*

Mallek, H., Ghozzi, F., Teste, O., & Gargouri, F. (2018). *BigDimETL with NoSQL Database. Procedia Computer Science, 126, 798-807.*

Yangui, R., Nabli, A., & Gargouri, F. (2017). *ETL Based Framework for NoSQL Warehousing. In M. Themistocleous & V. Morabito (Éds.), Information Systems (p. 40-53). Springer International Publishing.*